MyloBot 2022 – Evasive botnet that just sends extortion emails?

Natalie Zargarov

MyloBot was first detected in 2018 and was one of the most evasive Botnets at the time. According to various reports, it incorporated different techniques such as:

- Anti VM techniques
- Anti-sandbox techniques

- Anti-debugging techniques
- Wrapping internal parts with an encrypted resource file
- Code injection
- Process hollowing - a technique in which an attacker creates a new process in a suspended state, and later replaces that process's code with the the malicious one in order to remain undetected.
- Reflective EXE - executing EXE files directly from memory, without having them on disk.

- Delaying mechanism of 14 days before accessing its command-and-control servers.

We recently found a 2022 version of MyloBot and decided to check whether and how this sophisticated botnet has evolved and were quite surprised, as it would appear that not much has changed since 2018. Several Anti – Debugging and Anti – VM techniques have disappeared and more injection techniques are now being implemented but, ultimately, the second stage payload downloaded from the C&C server is used to send Extortion emails. But let's dig into the analysis and talk about the most interesting techniques used in our sample.

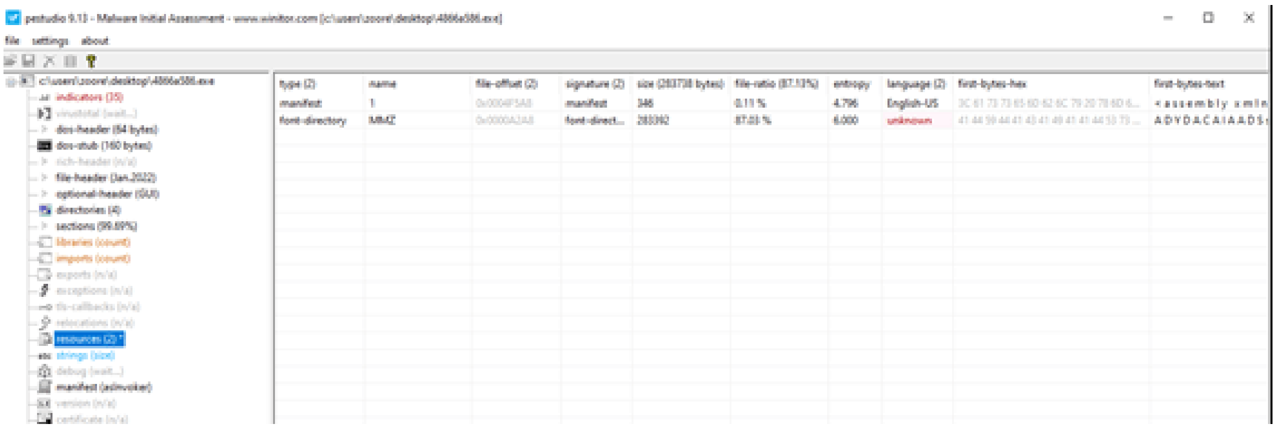
First Stage of the Attack

The first tricky Anti-Debugging technique used by the author is setting up an unhandled exception filter using “SetUnhandledExceptionFilter”. Usually, when an exception occurs and no exception handler is registered, the kernel32!UnhandledExceptionFilter() function is called. Using “SetUnHandledExceptionFilter” makes it possible to register a custom unhandled exception filter. But if the program is being debugged, the custom filter will not be called, and the exception will be passed to the debugger. This means that the threat actor can register its real payload in a custom exception filter and hide it from analyst eyes.

Another significant technique used in the first stage of the attack is a call to the “CreateTimerQueueTimer” WINAPI function. According to the documentation, this function “Creates a timer-queue timer. This timer expires at the specified due time, then after every specified period. When the timer expires, the callback function is called.” Which is exactly what is happening in our sample.

In the callback function the author triggers the UnhandledExceptionFilter by dividing by zero, which triggers the custom unhandled exception filter to be called.

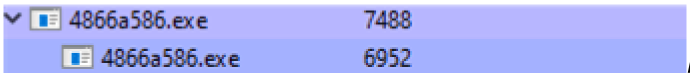
Later, after ensuring that it is not being debugged, MyloBot looks for the ‘MMZ’ resource, which is an encrypted executable. If the resource is found, it loads it into the memory and decodes it:



type (2)	name	file-offset (2)	signature (2)	size (283738 bytes)	file-ratio (87.13%)	entropy	language (2)	first-bytes-hex	first-bytes-text
manifest	1	0x0004F5A8	manifest	346	0.11 %	4.796	English-US	3C 61 73 73 65 6D 62 6C 79 20 78 6D 6...	< a s s e m b l y x m l n
font-directory	MMZ	0x0000A2A8	font-direct...	283392	87.03 %	6.000	unknown	41 44 59 44 41 43 41 49 41 41 44 53 73 ...	A D Y D A C A I A A D S

Figure 1 - First stage resources

The decoded executable is injected into a newly running in-suspended-state process. This process is an instance of the original executable whose memory was unmapped and replaced with the decoded resource file, a technique called Process Hollowing.

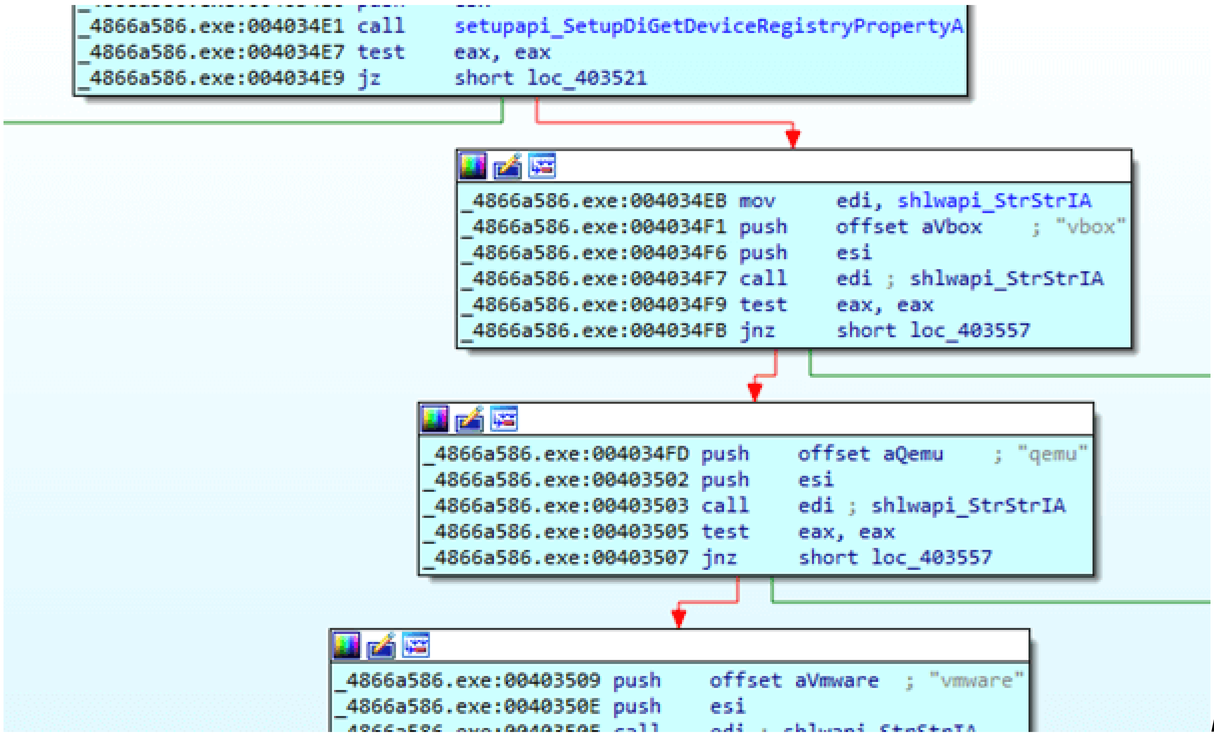


4866a586.exe	7488
4866a586.exe	6952

Figure 2 - Hollowed Process

Second Stage of the Attack – decoded resource

With the resuming thread of the second stage hollowed process it performs an Anti-VM check using SetupDiGetClassDevs, SetupDiEnumDeviceInfo and SetupDiGetDeviceRegistryProperty to query the friendly name of all devices present on the current system and checks for the the strings VMWARE, VBOX, VIRTUAL HD and QEMU within the name:



```
_4866a586.exe:004034E1 call    setupapi_SetupDiGetDeviceRegistryPropertyA
_4866a586.exe:004034E7 test    eax, eax
_4866a586.exe:004034E9 jz      short loc_403521
```

```
_4866a586.exe:004034EB mov     edi, shlwapi_StrStrIA
_4866a586.exe:004034F1 push    offset aVbox    ; "vbox"
_4866a586.exe:004034F6 push    esi
_4866a586.exe:004034F7 call    edi ; shlwapi_StrStrIA
_4866a586.exe:004034F9 test    eax, eax
_4866a586.exe:004034FB jnz     short loc_403557
```

```
_4866a586.exe:004034FD push    offset aQemu    ; "qemu"
_4866a586.exe:00403502 push    esi
_4866a586.exe:00403503 call    edi ; shlwapi_StrStrIA
_4866a586.exe:00403505 test    eax, eax
_4866a586.exe:00403507 jnz     short loc_403557
```

```
_4866a586.exe:00403509 push    offset aVmware  ; "vmware"
_4866a586.exe:0040350E push    esi
_4866a586.exe:0040350F call    edi ; shlwapi_StrStrIA
```

Figure 3 - Anti-VM Check

Next, it checks for persistent software/other persistent malware running on the endpoint by querying subkeys of:

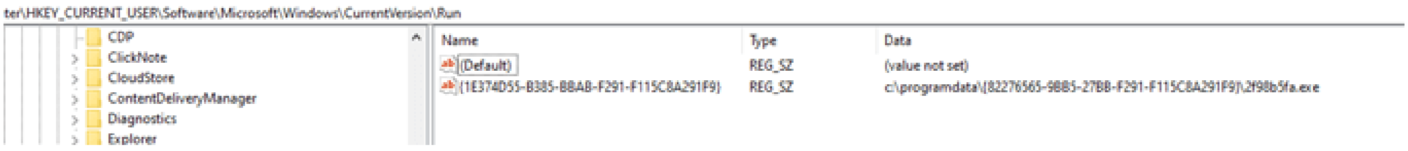
- HKCU\Software\Microsoft\Windows\CurrentVersion\Run
- HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce
- HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Run
- HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\RunOnce
- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce

If any subkey is found and that subkey name does not relate to “Microsoft”, the malware queries its value, checks if the value contains the ‘.exe’ extension, checks if the file exists and if it is currently running. If it is running, it terminates it. This is most likely in order to terminate other malware or utilities running on the endpoint.

The second stage executable then creates a new folder under C:\ProgramData . It looks for svchost.exe under a system directory and executes it in suspended state. Using an APC injection technique, it injects itself into the spawned svchost.exe process.

Third Stage of the Attack – svchost.exe

This stage is responsible for creating persistency. First, it writes the first stage executable to the created folder, after that, it creates a registry key under HKCU\Software\Microsoft\Windows\CurrentVersion\Run with the path to the written file as a value:



```
ter\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
```

Name	Type	Data
(Default)	REG_SZ	(value not set)
{1E374D55-B385-BBAB-F291-F115C8A291F9}	REG_SZ	c:\programdata\{82276565-9BB5-27BB-F291-F115C8A291F9}\2f98b5fa.exe

Figure 4 - Persistency

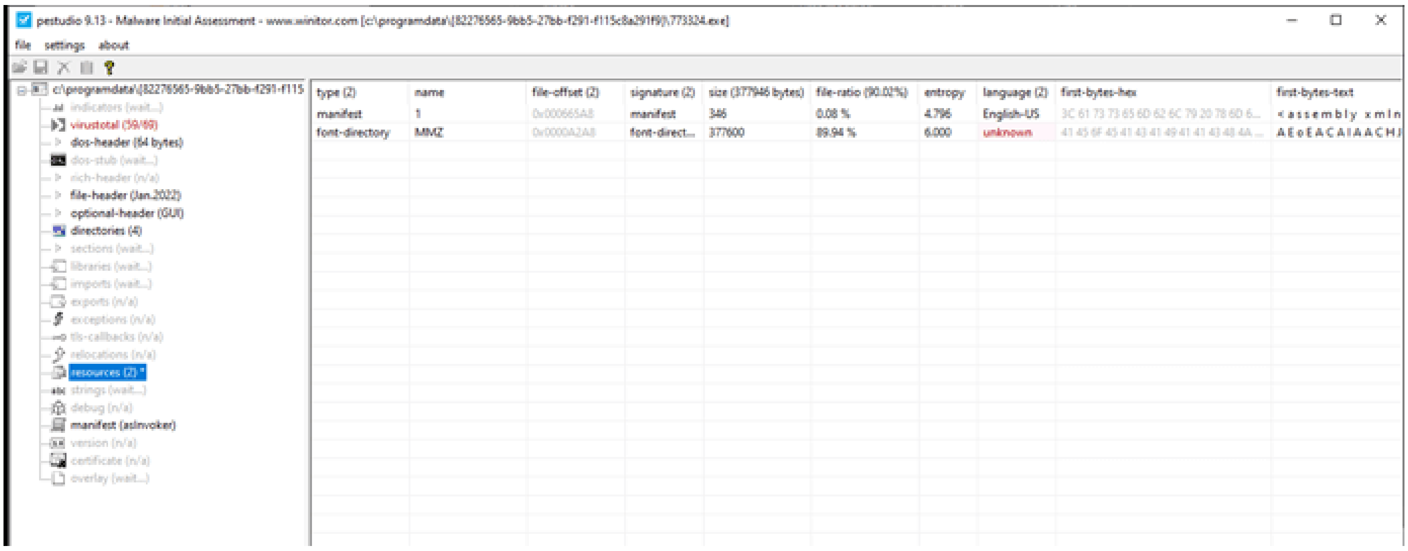
Next, it executes the written file. It checks running processes and looks for a 64bit process. When it finds it, it injects a 90-byte function into the process and a path to the written file. In this stage the injection is performed using the CreateRemoteThread technique. After some delay, the injected 64bit process function will execute the path that was written. If no 64bit processes were found, the process runs notepad.exe and injects into that.

Fourth Stage of the Attack

The file used in this stage is a copy of the first stage file. However, this time its purpose is to contact a C&C server, download the next stage executable (and an update.txt file in some cases) and execute it.

Fifth Stage of the Attack

When a file is downloaded and executed on the victim's endpoint, we see again the previously reviewed evasion patterns. It uses a "SetUnHandledExceptionFilter" as the Anti-Debugger, "CreateTimerQueueTimer" for a callback function execution, division by zero to trigger "UnhandledExceptionFilter" and encrypted resource loading and decrypting. One thing that is different in this stage is the resource itself, as we can see it is not the same as the previous:



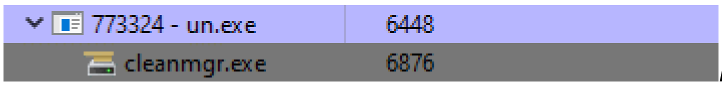
pestudio 9.13 - Malware Initial Assessment - www.winitor.com [c:\programdata\{82276565-9bb5-27bb-f291-f115c8a291f9}\773324.exe]

type (2)	name	file-offset (2)	signature (2)	size (377946 bytes)	file-ratio (90.02%)	entropy	language (2)	first-bytes-hex	first-bytes-text
manifest	1	0x000665A8	manifest	346	0.08 %	4.796	English-US	3C 61 73 73 65 6D 62 6C 79 20 78 6D 6...	< a s s e m b l y x m l n
font-directory	MMZ	0x0000A2A8	font-direct...	377600	89.94 %	6.000	unknown	41 45 6F 45 41 43 41 49 41 41 43 48 4A...	A E o E A C A I A A C H J

Figure 5 - Downloaded payload resources

It uses the previously mentioned Process Hollowing technique to inject the decoded code into a new process and executes it.

The Hollowed process first checks if a file named "update.txt" exists. If it does, it reads the file into the process memory. Then, it runs a 64bit cleanmgr.exe process and injects the whole payload into it using a CreateRemoteThread technique. If the cleanmgr.exe file does not exist, it uses a svchost.exe file instead:

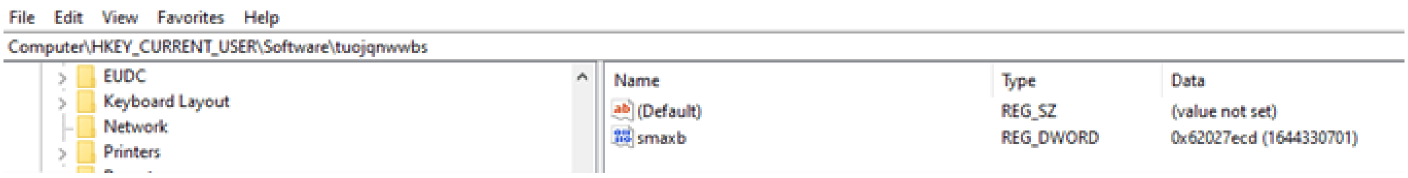


773324 - un.exe	6448
cleanmgr.exe	6876

Figure 6 - Injection into suspended cleanmgr.exe

Sixth Stage of the Attack – downloaded payload

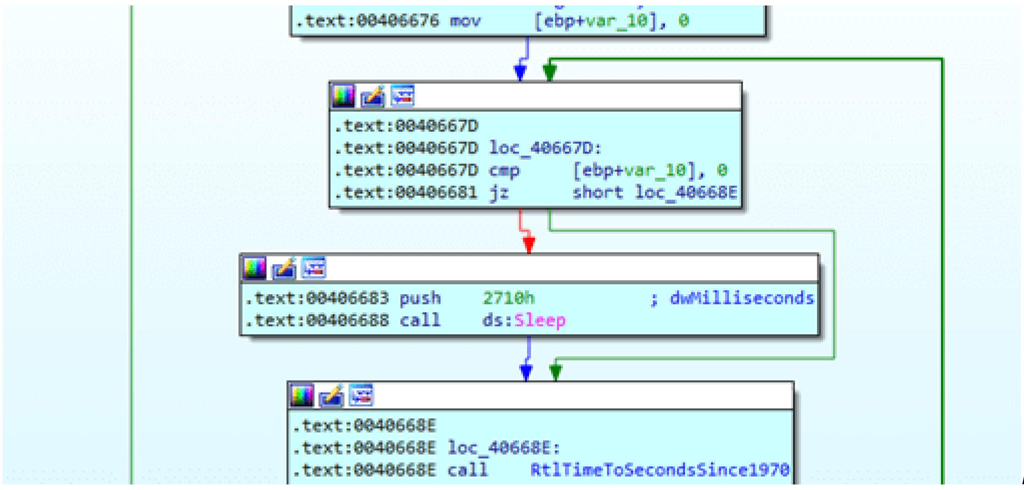
Running as cleanmgr.exe, a malicious payload first uses an additional Timing Anti-Debugging technique. When a process is traced in a debugger, there is a huge delay between instructions and execution. The "native" delay between some parts of code can be measured and compared with the actual delay using several approaches. Our sample creates a new HKCU\Software\tuojqnwwbs registry key, calls RtlTimeToSecondsSince1970 which is used for "Converting the specified 64-bit system time to the number of seconds since the beginning of January 1, 1970" and saves the value in the created registry key:


File Edit View Favorites Help
Computer\HKEY_CURRENT_USER\Software\tuojqnwwbs
EUDC
Keyboard Layout
Network
Printers

Name	Type	Data
(Default)	REG_SZ	(value not set)
smaxb	REG_DWORD	0x62027ecd (1644330701)

Figure 7 - Timing Anti-Debugging. Registry key

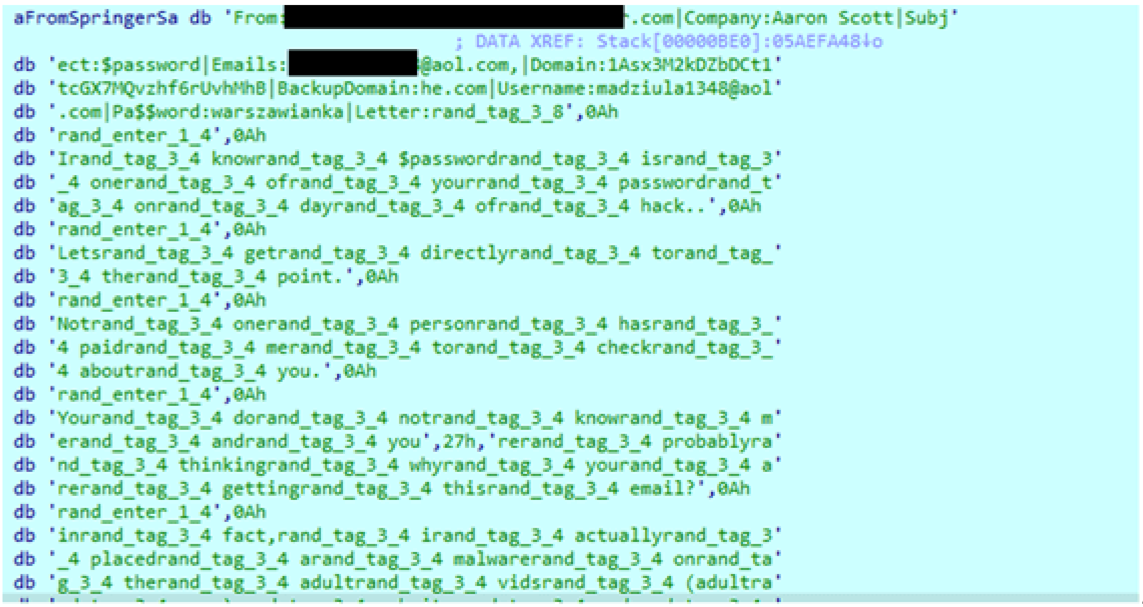
It calls RtlTimeToSecondsSince1970 again later and checks a time delta:



```
.text:00406676 mov     [ebp+var_10], 0

.text:0040667D
.text:0040667D loc_40667D:
.text:0040667D cmp     [ebp+var_10], 0
.text:00406681 jz      short loc_40668E

.text:00406683 push    2710h           ; dwMilliseconds
.text:00406688 call    ds:Sleep

.text:0040668E
.text:0040668E loc_40668E:
.text:0040668E call    RtlTimeToSecondsSince1970
```

Figure 8 - Part of Timing Anti-Debugging function

Next, a "\Sessions\2\BaseNamedObjects\Santiv18" mutex is created. If the "update.txt" file is not found, the malware uses the victim's endpoint to send Extortion emails:



```
aFromSpringerSa db 'From:[redacted].com|Company:Aaron Scott|Subj'
                                        ; DATA XREF: Stack[00000BE0]:05AEFA48↓o
db 'ect:$password|Emails:[redacted]@aol.com,|Domain:1Asx3M2kDZbDCt1'
db 'tcGX7MQvzhf6rUvhMh8|BackupDomain:he.com|Username:madziula1348@aol'
db '.com|Pa$$word:warszawianka|Letter:rand_tag_3_8',0Ah
db 'rand_enter_1_4',0Ah
db 'Irand_tag_3_4 knowrand_tag_3_4 $passwordrand_tag_3_4 isrand_tag_3'
db '_4 onerand_tag_3_4 ofrand_tag_3_4 yourrand_tag_3_4 passwordrand_t'
db 'ag_3_4 onrand_tag_3_4 dayrand_tag_3_4 ofrand_tag_3_4 hack..',0Ah
db 'rand_enter_1_4',0Ah
db 'Letsrand_tag_3_4 getrand_tag_3_4 directlyrand_tag_3_4 torand_tag_'
db '3_4 therand_tag_3_4 point.',0Ah
db 'rand_enter_1_4',0Ah
db 'Notrand_tag_3_4 onerand_tag_3_4 personrand_tag_3_4 hasrand_tag_3_'
db '4 paidrand_tag_3_4 merand_tag_3_4 torand_tag_3_4 checkrand_tag_3_'
db '4 aboutrand_tag_3_4 you.',0Ah
db 'rand_enter_1_4',0Ah
db 'Yourand_tag_3_4 dorand_tag_3_4 notrand_tag_3_4 knowrand_tag_3_4 m'
db 'erand_tag_3_4 andrand_tag_3_4 you',27h,'rerand_tag_3_4 probablyra'
db 'nd_tag_3_4 thinkingrand_tag_3_4 whyrand_tag_3_4 yourand_tag_3_4 a'
db 'rerand_tag_3_4 gettingrand_tag_3_4 thisrand_tag_3_4 email?',0Ah
db 'rand_enter_1_4',0Ah
db 'inrand_tag_3_4 fact,rand_tag_3_4 irand_tag_3_4 actuallyrand_tag_3'
db '_4 placedrand_tag_3_4 arand_tag_3_4 malwarerand_tag_3_4 onrand_ta'
db 'g_3_4 therand_tag_3_4 adultrand_tag_3_4 vidsrand_tag_3_4 (adultra'
```

Figure 9 - Extortion message

The email content is:

I know michigan is one of your password on day of hack..

Lets get directly to the point.

Not one person has paid me to check about you.

You do not know me and you're probably thinking why you are getting this email?

in fact, i actually placed a malware on the adult vids (adult porn) website and you know what, you visited this site to experience fun (you know what i mean).

When you were viewing videos, your browser started out operating as a RDP having a key logger which provided me with accessibility to your display and web cam.

immediately after that, my malware obtained every one of your contacts from your Messenger, FB, as well as email account.

after that i created a double-screen video. 1st part shows the video you were viewing (you have a nice taste omg), and 2nd part displays the recording of your cam, and its you.

Best solution would be to pay me $2732.

We are going to refer to it as a donation. in this situation, i most certainly will without delay remove your video.

My BTC address : 14JuDQdSEQtFq7SkFHGJackAxneY9ixAUM

[case SeNSiTiVe, copy & paste it]

You could go on your life like this never happened and you will not ever hear back again from me.

You'll make the payment via Bitcoin (if you do not know this, search 'how to buy bitcoin' in Google).

if you are planning on going to the law, surely, this e-mail can not be traced back to me, because it's hacked too.

I have taken care of my actions. i am not looking to ask you for a lot, i simply want to be paid.

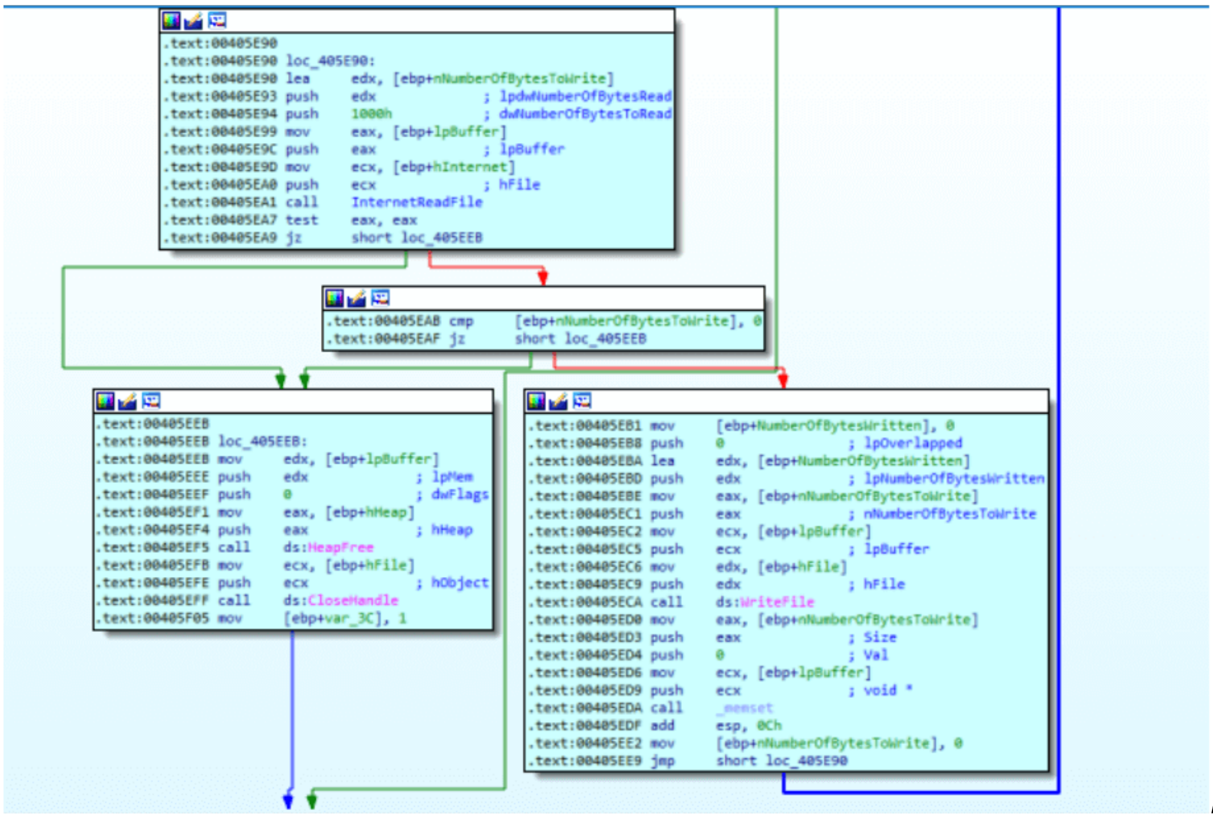
if i do not receive the bitcoin;, I definitely will send out your video recording to all of your contacts including friends and family, co-workers, and so on.

Nevertheless, if i do get paid, i will destroy the recording immediately.

If you need proof, reply with Yeah then i will send out your video recording to your 8 friends.

it's a nonnegotiable offer and thus please don't waste mine time & yours by replying to this message.

Even though this seems to be the final payload, we observed that it still has the ability to download an additional file to the endpoint:



```
.text:00405E90
.text:00405E90 loc_405E90:
.text:00405E90 lea     edx, [ebp+nNumberOfBytesToWrite]
.text:00405E93 push    edx             ; lpdwNumberOfBytesRead
.text:00405E94 push    1000h           ; dwNumberOfBytesToRead
.text:00405E99 mov     eax, [ebp+lpBuffer]
.text:00405E9C push    eax             ; lpBuffer
.text:00405E9D mov     ecx, [ebp+hInternet]
.text:00405EA0 push    ecx             ; hFile
.text:00405EA1 call    InternetReadFile
.text:00405EA7 test    eax, eax
.text:00405EA9 jz      short loc_405EEB
```

```
.text:00405EAB cmp     [ebp+nNumberOfBytesToWrite], 0
.text:00405EAF jz      short loc_405EEB
```

```
.text:00405EEB
.text:00405EEB loc_405EEB:
.text:00405EEB mov     edx, [ebp+lpBuffer]
.text:00405EEE push    edx             ; lpMem
.text:00405EEF push    0               ; dwFlags
.text:00405EF1 mov     eax, [ebp+hHeap]
.text:00405EF4 push    eax             ; hHeap
.text:00405EF5 call    ds:HeapFree
.text:00405EFB mov     ecx, [ebp+hFile]
.text:00405EFE push    ecx             ; hObject
.text:00405EFF call    ds:CloseHandle
.text:00405F05 mov     [ebp+var_3C], 1
```

```
.text:00405EB1 mov     [ebp+NumberOfBytesWritten], 0
.text:00405EB8 push    0               ; lpOverlapped
.text:00405EBA lea     edx, [ebp+NumberOfBytesWritten]
.text:00405EBD push    edx             ; lpNumberOfBytesWritten
.text:00405EBE mov     eax, [ebp+nNumberOfBytesToWrite]
.text:00405EC1 push    eax             ; nNumberOfBytesToWrite
.text:00405EC2 mov     ecx, [ebp+lpBuffer]
.text:00405EC5 push    ecx             ; lpBuffer
.text:00405EC6 mov     edx, [ebp+hFile]
.text:00405EC9 push    edx             ; hFile
.text:00405ECA call    ds:WriteFile
.text:00405ED0 mov     eax, [ebp+nNumberOfBytesToWrite]
.text:00405ED3 push    eax             ; Size
.text:00405ED4 push    0               ; Val
.text:00405ED6 mov     ecx, [ebp+lpBuffer]
.text:00405ED9 push    ecx             ; void *
.text:00405EDA call    _memset
.text:00405EDF add     esp, 0Ch
.text:00405EE2 mov     [ebp+nNumberOfBytesToWrite], 0
.text:00405EE9 jmp     short loc_405E90
```

Figure 10 - Downloading function

This might indicate that the threat actor left a door open for itself and might yet decide to pass additional files.

We also noticed that the payload has the ability to create a scheduled task which might be used for adding persistency for the next stage (if it is performed).

This threat actor went through a lot of trouble to drop the malware and keep it undetected, only to use it as an extortion mail sender. Botnets are dangerous exactly because of this unknown upcoming threat. It could just as easily drop and execute ransomware, spyware, worms, or other threats on all infected endpoints.

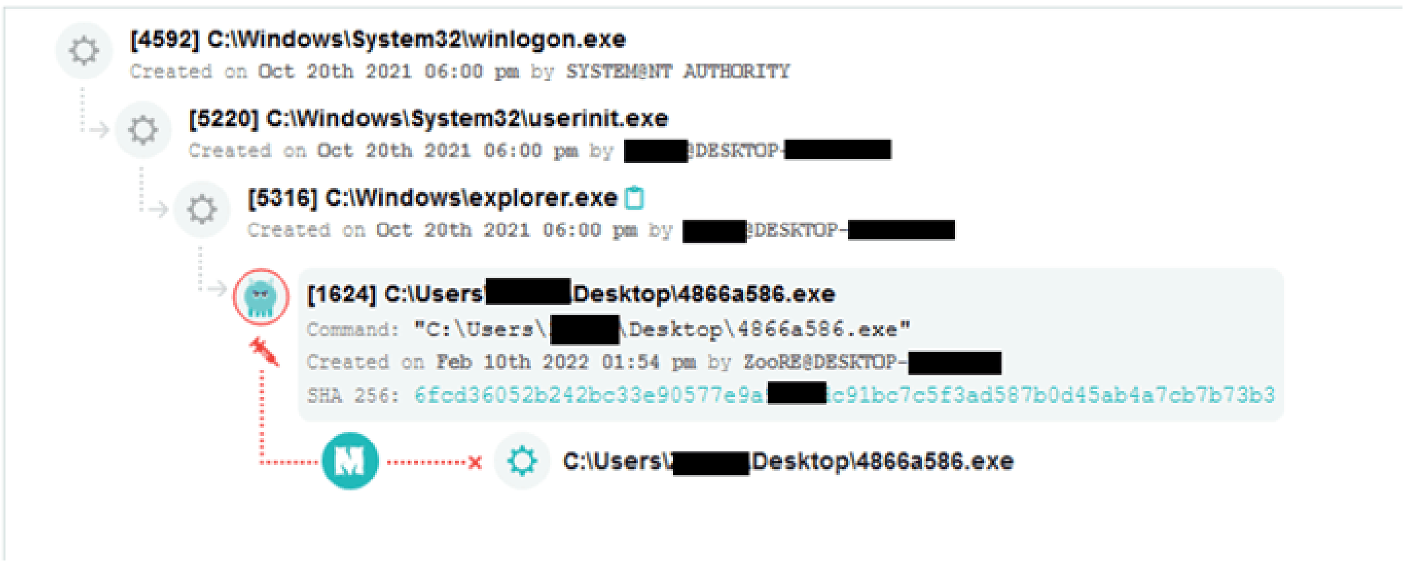
We observed a large amount of similar samples at VT.

This is an interesting example of an attack that might not appear to be “harmful” or very common, but it holds 2 major risks for an organization:

1. The organization domain might find itself block by other organizations which could in turn severely harm its
2. The attacker has gained a significant foothold on the network domain and has made harmless actions. But, after a while, the attacker may change the payload and impose more malicious actions, like in the examples above.

How Minerva Labs prevents these types of attacks

Minerva Labs Memory Injection Prevention module prevents this attack and protects the endpoint from becoming a bot:



```
[4592] C:\Windows\System32\winlogon.exe
Created on Oct 20th 2021 06:00 pm by SYSTEM@NT AUTHORITY
  [5220] C:\Windows\System32\userinit.exe
  Created on Oct 20th 2021 06:00 pm by [redacted]@DESKTOP-[redacted]
    [5316] C:\Windows\explorer.exe
    Created on Oct 20th 2021 06:00 pm by [redacted]@DESKTOP-[redacted]
      [1624] C:\Users[redacted]Desktop\4866a586.exe
      Command: "C:\Users\[redacted]\Desktop\4866a586.exe"
      Created on Feb 10th 2022 01:54 pm by ZooRE@DESKTOP-[redacted]
      SHA 256: 6fcd36052b242bc33e90577e9a[redacted]c91bc7c5f3ad587b0d45ab4a7cb7b73b3
        C:\Users\[redacted]Desktop\4866a586.exe
```

IOC’s

First stage executable - 4866a586.exe
6fcd36052b242bc33e90577e9a9cf5dc91bc7c5f3ad587b0d45ab4a7cb7b73b3